

Asynchronous Serial Interfaces and the Internet of Things

Lecture 12

Josh Brake
Harvey Mudd College

Outline

- Serial Interfaces pt. 2 – the Universal Synchronous/Asynchronous Receiver Transmitter
- General Internet Architecture
 - Protocol layers
 - Browsing the Web
 - HTTP - Commands and Format
 - HTML - Hypertext Markup Language
- ESP8266
 - Overview
 - Webserver Code
 - Basic workflow for whole system

Learning Objectives

By the end of this lecture you should be able to...

- Articulate the differences and tradeoffs between a synchronous serial link (e.g., SPI) and an asynchronous serial link.
- Use the USART peripheral on the MCU to print to the terminal window
- Write a basic HTML webpage.
- Explain the basic operating principles of an HTTP webserver.

Universal Synchronous/Asynchronous Receiver Transmitter (USART)

What if we don't want a shared clock?

We must...

- Agree on shared data rate
- Sample the incoming data stream at higher frequency to synchronize the input data stream with the reading circuitry
- Add additional bits at the beginning and end of the transmission to signal the bounds of the transmission

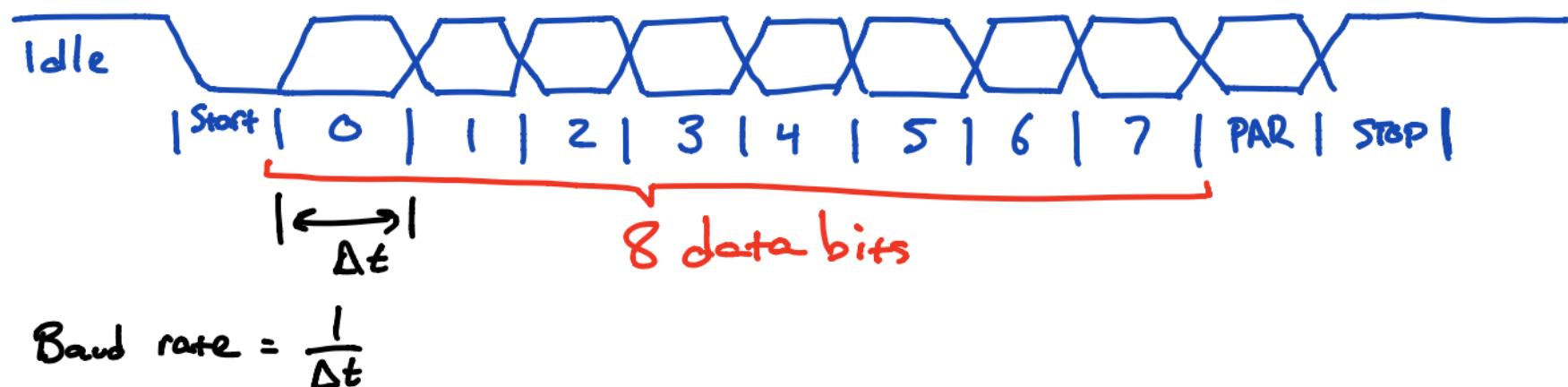
Q: What are some downsides of an asynchronous serial interface as compared to a synchronous one?

- Reduced **maximum transmission frequency** (typically 8x-16x overhead from sampling)
- Wasted **bits** in each transmission

USART Data Frame

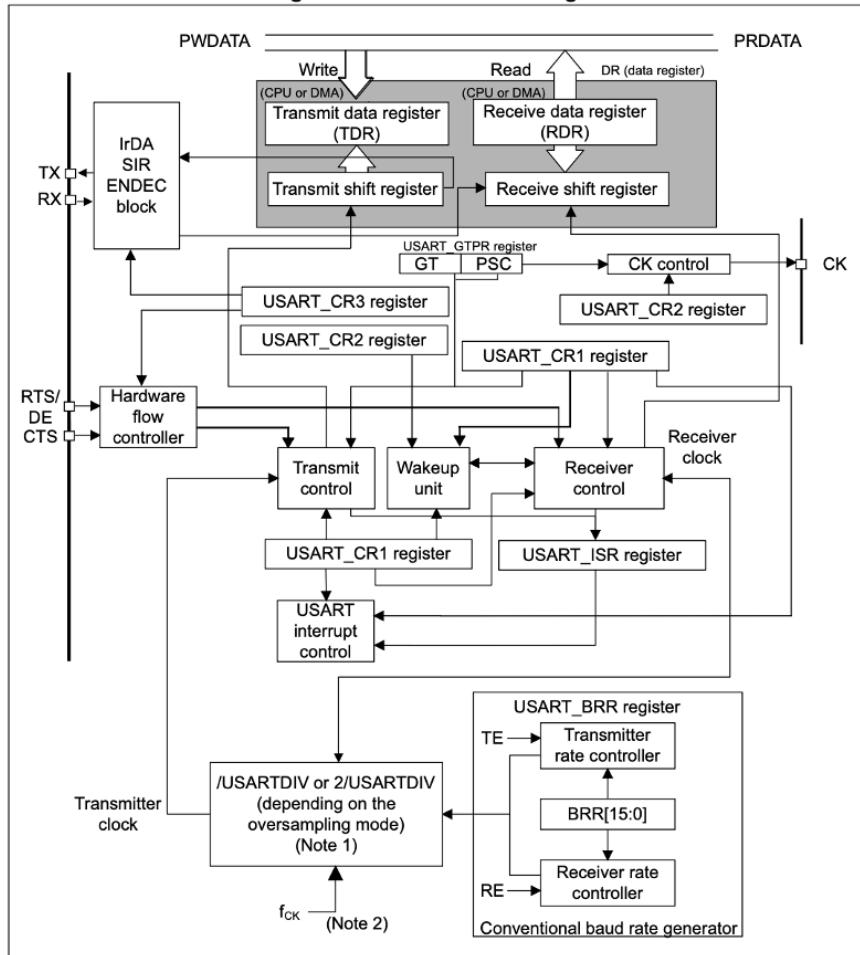
4 components

1. Start bit: always logical 0
2. Data bits: 5-9 bits of data
3. Parity bit: Option bit with parity of data (i.e., even or odd. Simple error checking)
4. Stop bit(s): 1-2 bits. Always logical 1.



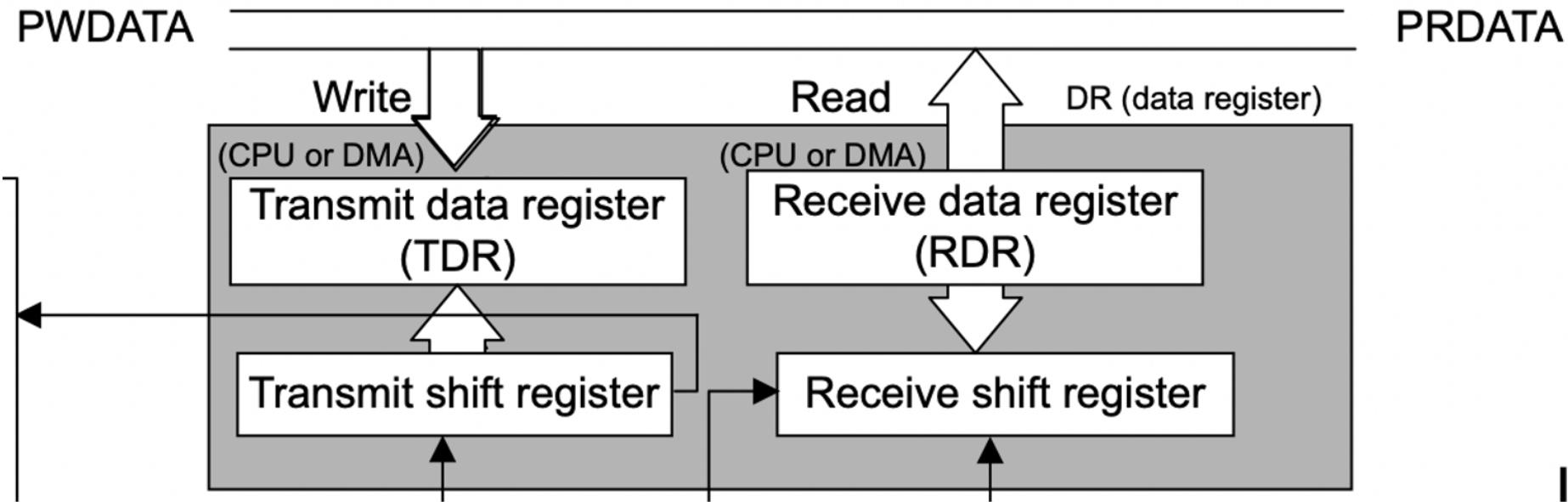
STM32L432KC USART

Figure 382. USART block diagram



RM0394 p. 1198

Data Registers



RM0394 p. 1198

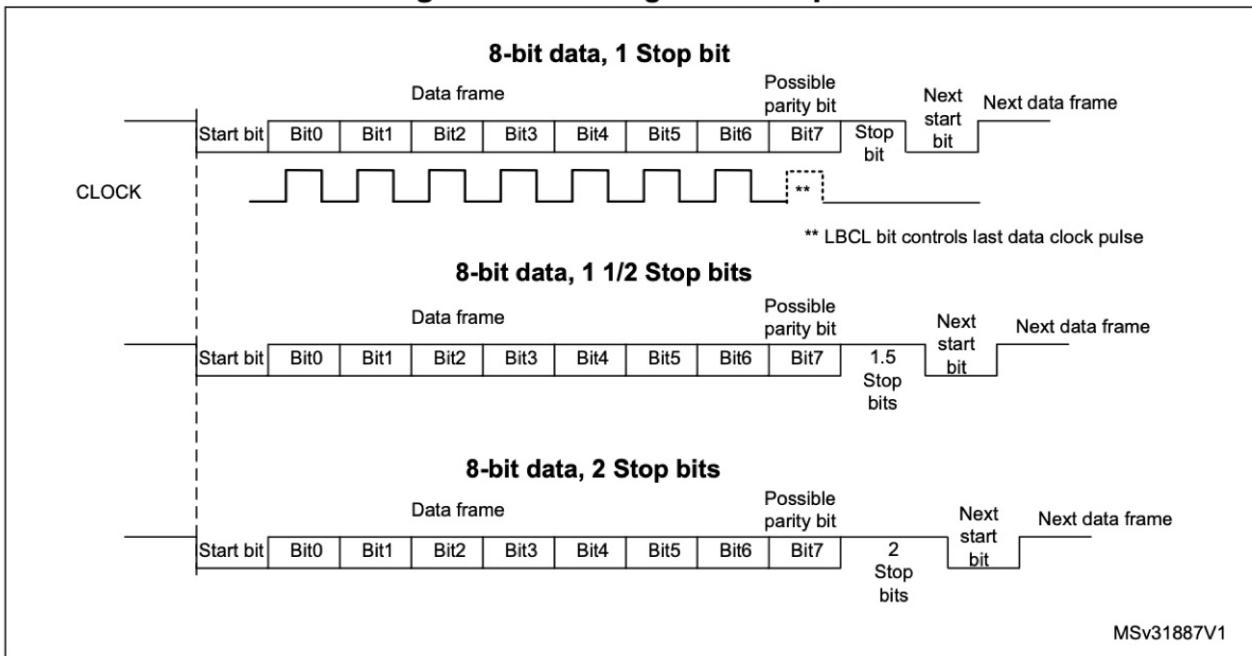
Pins

- TX – transmitted data out from USART
- RX – received data in to USART
- CK – (optional) clock output for synchronous mode
- RTS – Request To Send indicates the USART is ready to receive data (when low)
- CTS – Clear To Send block data transmission at the end of the current transfer when high

Data framing

Data framing

Figure 384. Configurable stop bits



RM0394 p. 1202

9

RM0394 p. 1202

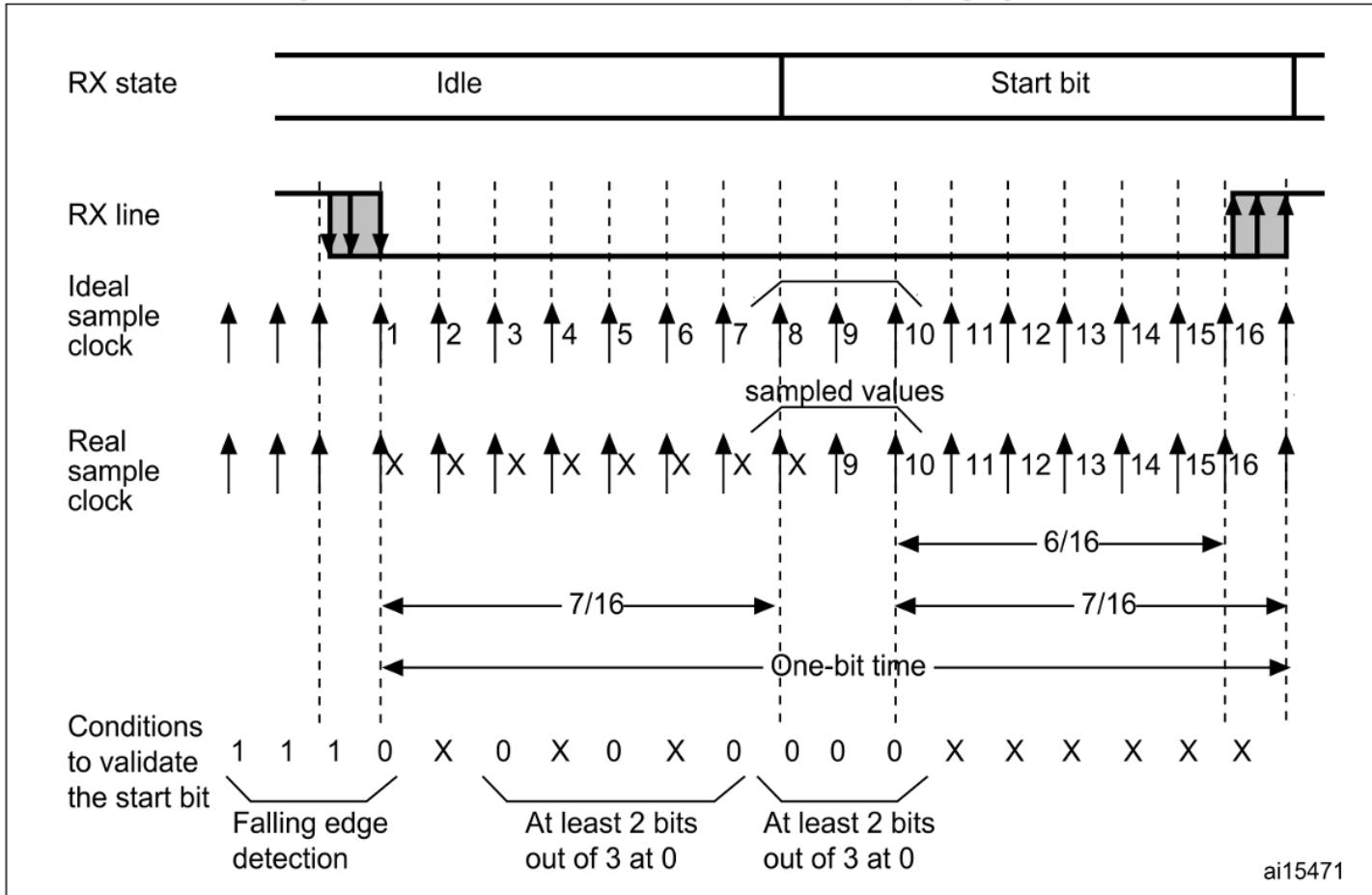
10 / 45

Error Flags

- Overrun – new byte in the **holding reg** before the old one was read out
- Frame – didn't get the **stop bit(s)** we expected
- Parity – calculated **parity** doesn't match **parity bit**.

Receiver

Figure 386. Start bit detection when oversampling by 16 or 8



RM0394 p. 1204

USART registers: Interrupt and Status Register (ISR)

UART Status Register

- **TXE** – transmit data register empty (0 if data is not transferred to the shift register, 1 if it is)
- **TC** – transmission complete flag
- **RXNE** – read data register not empty (0 if data has not been received, 1 if it is ready to be read)
- **FE** – framing error
- **PE** – parity error

38.8.8 Interrupt and status register (USART_ISR)

Address offset: 0x1C

Reset value: 0x0200 00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	TCBGT	Res.	Res.	REACK	TEACK	WUF	RWU	SBKF	CMF	BUSY
					r				r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRF	ABRE	Res.	EOBF	RTOF	CTS	CTSIF	LBDF	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
r	r		r	r	r	r	r	r	r	r	r	r	r	r	r

USART registers: Data Register

- Used for both reads and writes
- Max 9-bit data value DR [8:0]

USART registers: Baud Rate Register

38.8.4 Baud rate register (USART_BRR)

This register can only be written when the USART is disabled (UE=0). It may be automatically updated by hardware in auto baud rate detection mode.

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BRR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:4 **BRR[15:4]**

$\text{BRR}[15:4] = \text{USARTDIV}[15:4]$

Bits 3:0 **BRR[3:0]**

When $\text{OVER8} = 0$, $\text{BRR}[3:0] = \text{USARTDIV}[3:0]$.

When $\text{OVER8} = 1$:

$\text{BRR}[2:0] = \text{USARTDIV}[3:0]$ shifted 1 bit to the right.

$\text{BRR}[3]$ must be kept cleared.

USART registers: Control register 1

- **M**: word length 8 or 9 data bits
- **PCE**: parity control enable
- **TE**: transmitter enable
- **RE**: receiver enable

USART registers: Control register 2

- **STOP**: 2-bit field, number of stop bits (0.5, 1, or 2)
- Various clock control (if using in synchronous mode)

38.8.2 Control register 2 (USART_CR2)

Address offset: 0x04

Reset value: 0x0000 0000

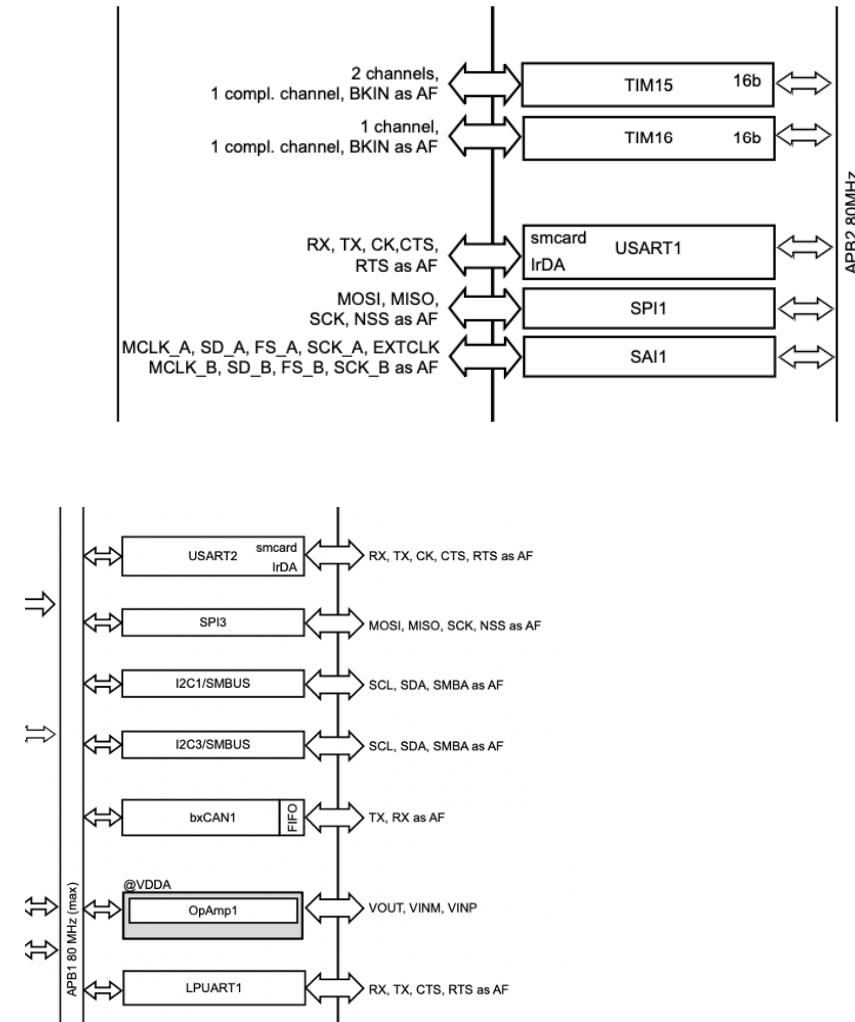
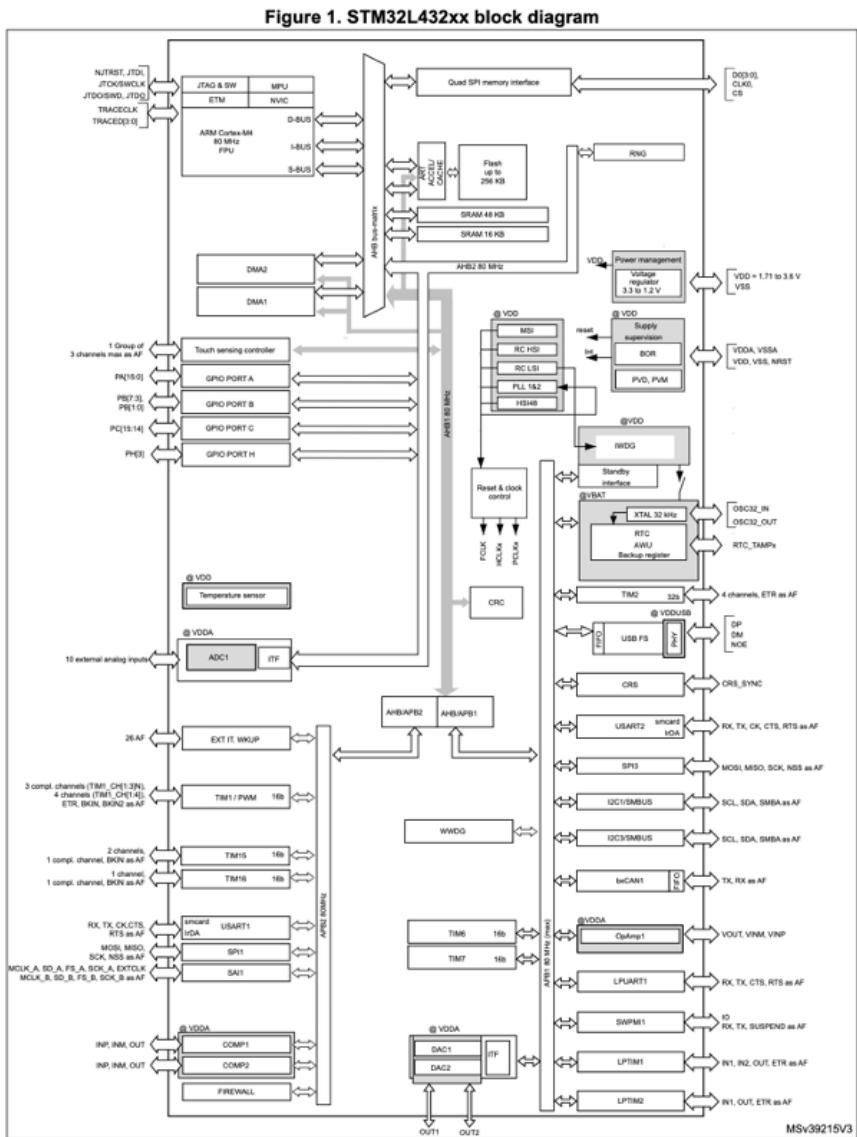
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
			ADD[7:4]			ADD[3:0]		RTOEN		ABRMOD[1:0]	ABREN	MSBFIRST	DATAINV	TXINV	RXINV
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWAP	LINEN		STOP[1:0]	CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	ADDM7	Res.	Res.	Res.	Res.
rw	rw		rw	rw	rw	rw	rw		rw	rw	rw				

Character Reception

1. Program the M bit in USART_CR1 to define word length
2. Program the sampling rate (x8 or x16) in USART_CR1
3. Program the number of stop bits in USART_CR2
4. (optional): Enable DMA
5. Select the desired baud rate in USART_BRR
6. Enable the USART with UE=1 in USART_CR1
7. Set the RE bit in USART_CR1

Wait for RXNE bit to go from 0 (no data received) to 1 (data received). Then, read out the data from the data register

USART Instances



USART Activity

Activity

Configure the USART as an UART to transmit serial data

- Read user manual and develop a bullet list outline of how to configure the peripheral
- Write USART library
- Finish STM32L432KC_USART.h and STM32L432KC_USART.c.
- Configure in common 8N1 mode
 - 8 data bits
 - No parity bit
 - 1 stop bit
 - Operate at 9600 baud (9.6 Kbps)
 - USART is configured to use the HSI which is 16 MHz.
- Use simple main function to transmit a string of your choice over the UART.

Bits to configure

38.8.1 Control register 1 (USART_CR1)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	M1	EOBIE	RTOIE	DEAT[4:0]					DEDT[4:0]				
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	CMIE	MME	M0	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	UESM	UE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- UE: USART Enable
- M: Word Length
- OVER8: Oversampling mode
- TE: Transmitter Enable
- RE: Receiver Enable (In CR2)
- STOP: Number of stop bits

Setup

- Download source code from GitHub
- Create new SEGGER project
- Configure serial monitor to read at 9600 baud

Table 15. Alternate function AF0 to AF7⁽¹⁾

Port	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
	SYS_AF	TIM1/TIM2/ LPTIM1	TIM1/TIM2	USART2	I2C1/I2C2/I2C3	SPI1/SPI2	SPI3	USART1/ USART2/ USART3
Port A	PA0	-	TIM2_CH1	-	-	-	-	USART2_CTS
	PA1	-	TIM2_CH2	-	-	I2C1_SMBA	SPI1_SCK	-
	PA2	-	TIM2_CH3	-	-	-	-	USART2_TX
	PA3	-	TIM2_CH4	-	-	-	-	USART2_RX
	PA4	-	-	-	-	SPI1_NSS	SPI3_NSS	USART2_CK
	PA5	-	TIM2_CH1	TIM2_ETR	-	-	SPI1_SCK	-
	PA6	-	TIM1_BKIN	-	-	SPI1_MISO	COMP1_OUT	USART3_CTS
	PA7	-	TIM1_CH1N	-	-	I2C3_SCL	SPI1_MOSI	-
	PA8	MCO	TIM1_CH1	-	-	-	-	USART1_CK
	PA9	-	TIM1_CH2	-	-	I2C1_SCL	-	USART1_TX
	PA10	-	TIM1_CH3	-	-	I2C1_SDA	-	USART1_RX
	PA11	-	TIM1_CH4	TIM1_BKIN2	-	-	SPI1_MISO	COMP1_OUT
	PA12	-	TIM1_ETR	-	-	-	SPI1_MOSI	-
	PA13	JTMS-SWDIO	IR_OUT	-	-	-	-	-
	PA14	JTCK-SWCLK	LPTIM1_OUT	-	-	I2C1_SMBA	-	-
	PA15	JTDI	TIM2_CH1	TIM2_ETR	USART2_RX	-	SPI1_NSS	SPI3_NSS
								USART3_RTS_DE

USART2 Wiring on Nucleo-32

6.9 USART virtual communication

Thanks to SB2 and SB3, the USART interface of STM32 available on PA2 (TX) and PA15 (RX), can be connected to ST-LINK/V2-1. When USART is not used it is possible to use PA2 as Arduino Nano A7. Refer to [Table 7](#).

Table 7. Virtual communication configuration

Bridge	State⁽¹⁾	Description
SB2	OFF	PA2 is connected to CN4 pin 5 as Arduino Nano analog input A7 and disconnected from ST-LINK USART.
	ON	PA2 is connected to ST-LINK as virtual Com TX (default).
SB3	OFF	PA15 is not connected.
	ON	PA15 is connected to ST-LINK as virtual Com RX (default).

1. The default configuration is reported in bold style.

USART2 Wiring

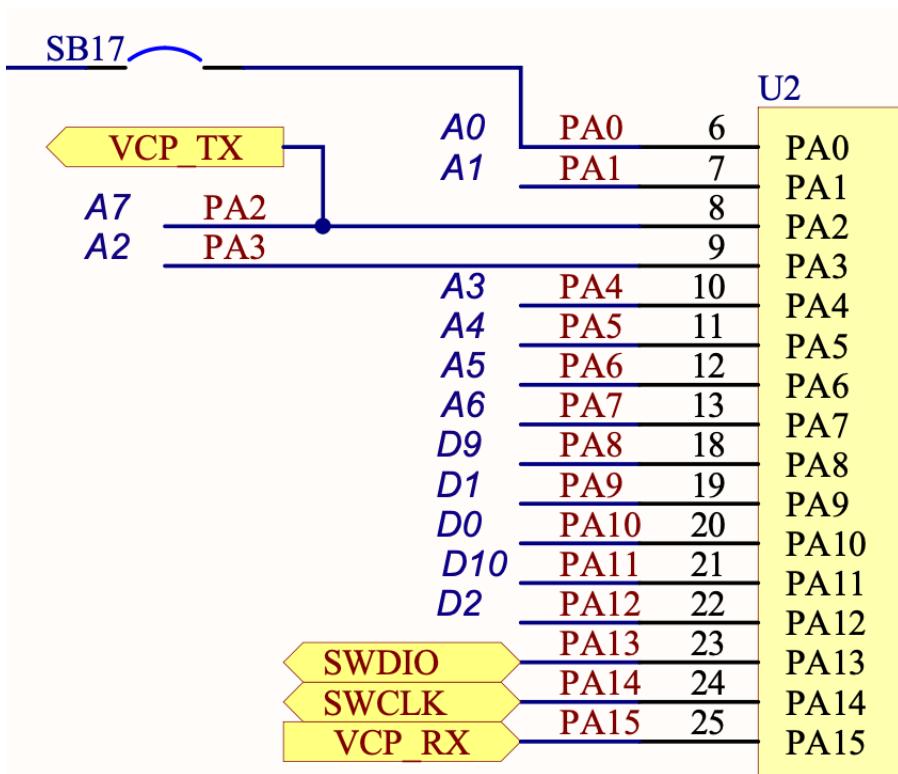


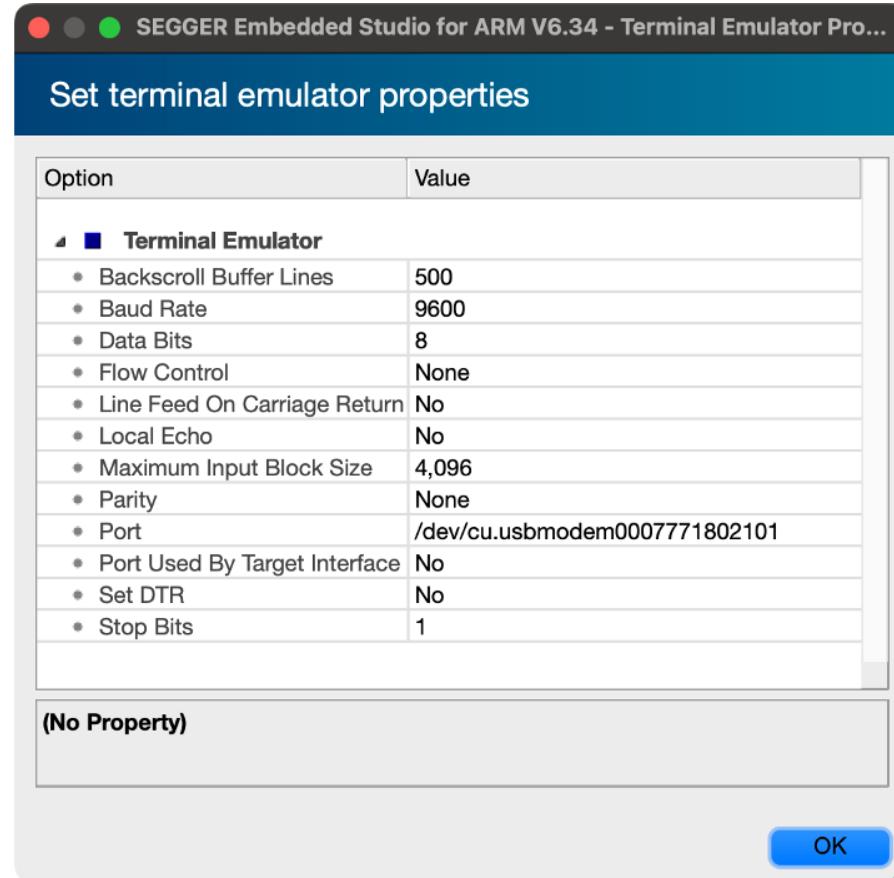
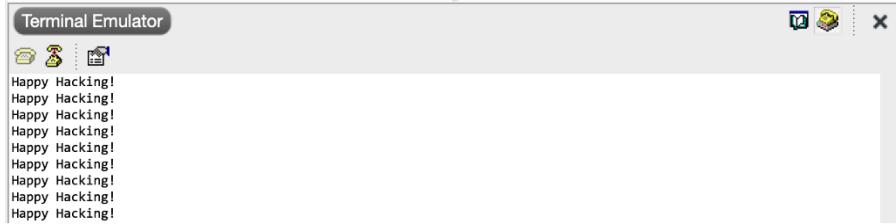
Figure 8. NUCLEO-L011K4, NUCLEO-L031K6, NUCLEO-L412KB and NUCLEO-L432KC pin assignment

NUCLEO-LxxxKx			
PA9	1	D1	VIN
PA10	2	D0	GND
NRST	3	NRST	NRST
GND	4	GND	+5V
PA12	5	D2	A7
PB0	6	D3	A6
PB7	7	D4	A5
PB6	8	D5	A4
PB1	9	D6	A3
PC14	10	D7	A2
PC15	11	D8	A1
PA8	12	D9	A0
PA11	13	D10	AREF
PB5	14	D11	+3V3
PB4	15	D12	D13
			15
			Arduino
			MSv40023V1

UM1956 p. 33

Receiving Serial Input over USB

Use built-in serial monitor in SES



Solution

```
1 ...
2 // Set M = 00
3 // M=00 corresponds to 1 start bit, 8 data bits, n stop bits
4 USART->CR1 &= ~(USART_CR1_M0 | USART_CR1_M1);
5 // Set to 16 times sampling freq
6 USART->CR1 &= ~USART_CR1_OVER8;
7 // 0b00 corresponds to 1 stop bit
8 USART->CR2 &= ~USART_CR2_STOP;
9
10
11 // Set baud rate to 115200 (see RM 38.5.4 for details)
12 // Tx/Rx baud = f_CK/USARTDIV (since oversampling by 16)
13 // f_CK = 16 MHz (HSI)
14
15 USART->BRR = (uint16_t) (HSI_FREQ / baud_rate);
16 // Enable USART
17 USART->CR1 |= USART_CR1_UE;
18 // Enable transmission and reception
19 USART->CR1 |= USART_CR1_TE | USART_CR1_RE;
20
21 return USART;
22 }
```

Solution

```
1 void sendChar(USART_TypeDef * USART, char data){  
2     while(!(USART->ISR & USART_ISR_TXE));  
3     USART->TDR = data;  
4     while(!(USART->ISR & USART_ISR_TC));  
5 }
```

Solution

```
1 // Lecture 12 Demo
2 // Josh Brake
3 // jbrake@hmc.edu
4 // 10/5/22
5
6 #include "STM32L432KC.h"
7 #include <stm32l432xx.h>
8 #define USART_ID USART2_ID
9 #define TIM TIM15
10
11 int main(void) {
12 // Configure flash and clock
13 configureFlash();
14 configureClock();
15
16 ...
```

Solution

```
1 ...
2 // Initialize USART
3 USART_TypeDef * USART = initUSART(USART_ID, 9600);
4
5 // Initialize timer
6 RCC->APB2ENR |= RCC_APB2ENR_TIM15EN;
7 initTIM(TIM);
8
9 char msg[28] = "Happy Hacking!\n\r";
10
11 while(1){
12     int i = 0;
13     do {
14         sendChar(USART, msg[i]);
15         i += 1;
16     } while (msg[i]);
17     delay_millis(TIM, 2000);
18 }
19 }
```

The Hypertext Transfer Protocol (HTTP)

Protocol Layers

IP - Internet Protocol Address



TCP: Transmission Control Prot.
IP: Internet Protocol
Hardware: Network card, modem, etc.

- Worldwide web is a service on the Internet
- Uses Hypertext Transfer Protocol (HTTP)
 - What layer is this protocol at?
- URL: Uniform Resource Locator
 - URL format: <protocol>://<hostname>:<port>/<path_and_filename>

Browsing the Web

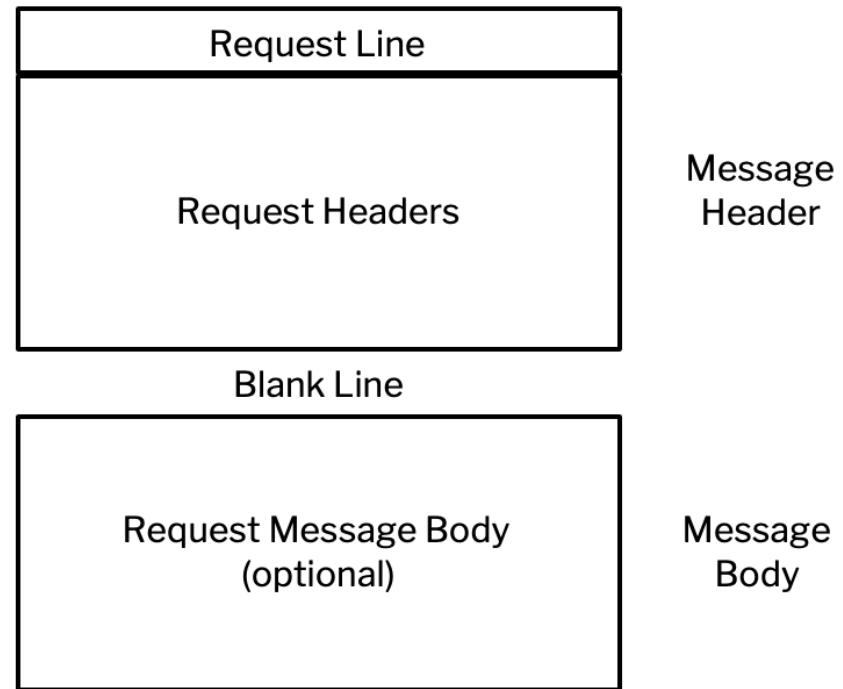
What happens when you type in a URL?

- Finds IP for domain if necessary (Using Dynamic Nameserver (DNS))
- Connects to server, send HTTP request
- Server receives request, searches for desired page.
 - If it exists, sends it.
 - If not, sends 404 “Page Not Found” error code.
- Web browser gets page, closes connection
- Parses webpage sending HTTP requests as necessary to get all the elements

HTTP: Commands and Format

GET

- Most common
- Used to request a resource
- Format
 - `GET / HTTP/1.1 Host: Accept`



HTML: HyperText Markup Language

Simple text format to specify webpage formatting

- Elements
 - DOCTYPE statement
 - HTML tag
 - Head
 - Body
- Tags look like `<tag>...</tag>`
- Common tags: `html, head, body, p, h<x> x={1,2,3}, title`

Activity: Simple HTML Page

- Open text editor (e.g., VSCode)
- Save document as .html
- Create example webpage below
- Open in web browser

```
1 <!DOCTYPE html>
2 <head>
3   <title>My First Webpage</title>
4 </head>
5 <body>
6   <h1>E155 Demo</h1>
7   <p>Put text here!</p>
8 </body>
```

Other HTML Elements

- Other HTML elements
 - Form
 - Attributes
 - type - submit
 - action - where to send form data
 - value - text on button
 - Add form to webpage

```
1 <form action="action_key">
2   <input type="submit" value="Send GET request">
3 </form>
```

ESP8266 Overview and Demo

Overview

ESP-WROOM-02 carries ESP8266EX highly integrated Wi-Fi SoC solution to meet the continuous demands for efficient power usage, compact design and reliable performance in the industry.

With the complete and self-contained Wi-Fi networking capabilities, it can perform as either a standalone application (WROOM module itself) or the slave to an MCU host which is the primary intention of the click board as a whole. So, this click board is applied to any microcontroller design as a Wi-Fi adaptor through UART interface (RX,TX lines on mikroBUS pin socket).

Notes	Pin	mikroBUS				Pin	Notes
	NC	1	AN	PWM	16	NC	
HW Reset	RST	2	RST	INT	15	NC	
Chip enable (active high)	EN	3	CS	TX	14	TX	UART0_TXD / Transmit end in UART download (program) mode
	NC	4	SCK	RX	13	RX	UART0_RXD / Receive end in UART download (program) mode
	NC	5	MISO	SCL	12	NC	
	NC	6	MOSI	SDA	11	NC	
Power supply	+3.3V	7	3.3V	5V	10	NC	
Ground	GND	8	GND	GND	9	GND	Ground

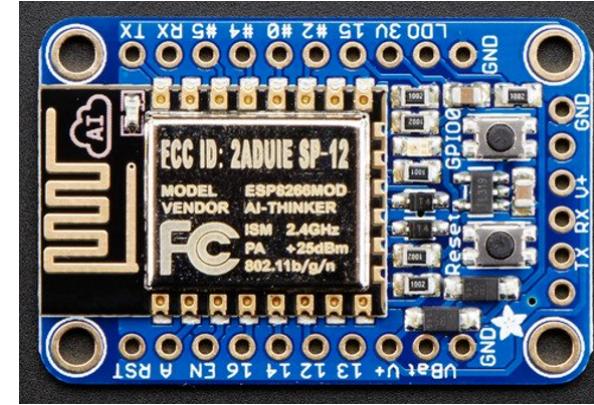


Overview

The Adafruit HUZZAH ESP8266 breakout is what we designed to make working with this chip super easy and a lot of fun. We took a certified module with an onboard antenna, and plenty of pins, and soldered it onto our designed breakout PCBs. We added in: - Reset button, - User button that can also put the chip into bootloading mode, - Red LED you can blink, - Level shifting on the UART and reset pin, - 3.3V out, 500mA regulator (you'll want to assume the ESP8266 can draw up to ~ 250mA so budget accordingly) - Two diode-protected power inputs (one for a USB cable, another for a battery)

Two parallel, breadboard-friendly breakouts on either side give you access to:

- 1 x Analog input (1.0V max)
- 9 x GPIO (3.3V logic), which can also be used for I2C or SPI
- 2 x UART pins
- 2 x 3-6V power inputs, reset, enable, LDO-disable, 3.3V output



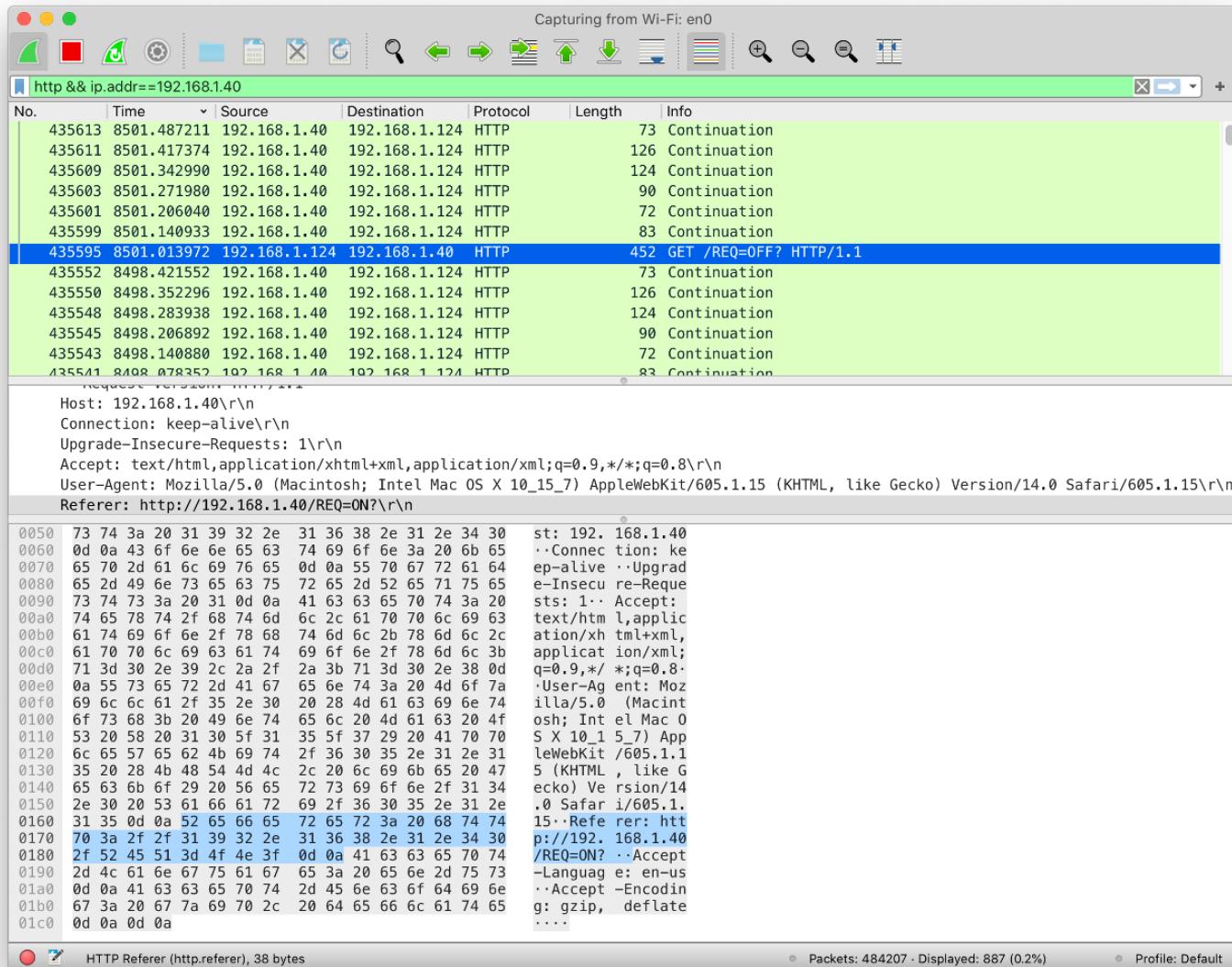
<https://cdn.sparkfun.com/datasheets/Wireless/WiFi/ESP8266ModuleV1.pdf>

ESP8266 Webserver Code

- Polls for waiting for a client to send an HTTP request
- When a request has been received, parses the request to slice the request after the /REQ: tag.
- Send the tag to the MCU which then decides what to do with the information.
- Then the MCU sends the content of the webpage back to the ESP8266 over the UART as properly formatted HTML.

ESP8266 Demo

Wireshark



Wrapup

- UART is a serial interface without a shared clock. Saves a wire, but at the cost of much slower data rates due to sampling overhead.
- Webpages in HTML are served using HTTP – sending text over a serial connection.